An introduction to Mesh generation

PhD Course: "An Introduction to DGFEM for partial differential equations"

Technical University of Denmark

August 25, 2009

Course content

The following topics are covered in the course

1. Introduction & DG-FEM in one spatial dimension
2. Implementation and numerical aspects (1D)
3. Insight through theory
4. Nonlinear problems
5. Extensions to two spatial dimensions
6. Introduction to mesh generation
7. Higher-order operators
8. Problem with three spatial dimensions and other advanced topics

Numerical solution of PDEs

To construct a numerical method for solving PDEs we need to consider

- How to represent the solution $u(x, t)$ by an approximate solution $u_h(x, t)$?
- In which sense will the approximate solution $u_h(x, t)$ satisfy the PDE?

The two choices separate and define the properties of different numerical methods...

The choice of how to represent the solution is intimately connected with the need for meshes

When do we need a mesh?

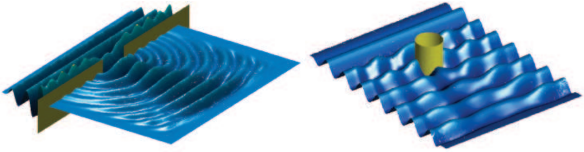
A mesh is needed when

- We want to solve a given problem on a computer using a method which requires a discrete representation of the domain

Two main problems to consider

- Numerical method:
  Which numerical method(s) to employ for defining a suitable solution procedure for the problem.
- Mesh generation:
  How to represent the domain of interest for use in our solution procedure.

When do we need a mesh?



It is convenient if

- We can independently consider the *problem solution procedure* and *mesh generation* as two distinct problems.

Domain of interest

In DG-FEM we have chosen to represent the problem domain Ω by a partitioning of the domain into a union of K nonoverlapping local elements D^k, $k = 1, ..., K$ such that

$$\Omega \cong \Omega_h = \bigcup_{k=1}^{K} D^k$$

Thus, the local representation of our solution u_h^k is intimately connected with the representation of the elements of the mesh.

For our purposes we are interested in nonoverlapping meshes...

What defines a mesh?

- A mesh is defined as a discrete representation Ω_h of some spatial domain Ω.
- A domain can be subdivided into K smaller non-overlapping closed subdomains Ω_h^k. The mesh is the union of such subdomains

$$\Omega_h = \bigcup_{k=1}^{K} \Omega_h^k$$

- The most common types of subdomains are polygons such as triangles/tehedra and quadrilaterals/cubes.
- Mesh generation can be a demanding and non-trivial task, especially for complex geometries.
- Easier to adapt unstructured meshes to practical problems in multi-dimensions involving complex geometry.

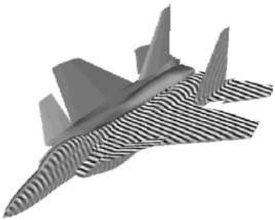


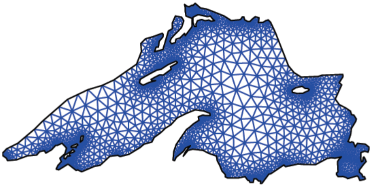
Figure: F-15. From www.useme.org

What defines a mesh?

Mesh terminology:

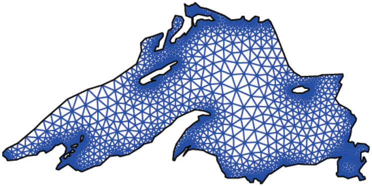
- Structured mesh
  Nearly all nodes have the same number of neighbors (interior vs. boundary nodes).
- Unstructured mesh
  Non-obvious number of neighbors for each node in mesh.
- Conformal mesh
  Nodes, sides and faces of neigboring elements are perfectly matched.
- Hanging nodes
  Nodes, which are not perfectly matched with a neighboring element node.

What defines a mesh? I



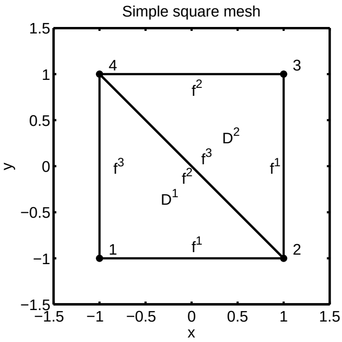
- A mesh is completely defined in terms of a set of (unique) vertices and defined connections among these.
  - coordinate tables, VX and VY (unique vertices)
  - mesh element table, EToV (triangulation/quadrilaterals/etc.)
- In addition it is customary to define types of boundaries for specifiying boundary conditions where needed.
  - a boundary type table, BCType (element face types)

What defines a mesh? I



- Very simple meshes can be created manually by hand.
- Automatic mesh generation is generally faster and more efficient
  - Some user input for accurately describing the geometry and desired (initial) mesh resolution may be required.
- Note: Mesh data can be stored for reuse several times
  - not necessary to generate every time!

Mesh data

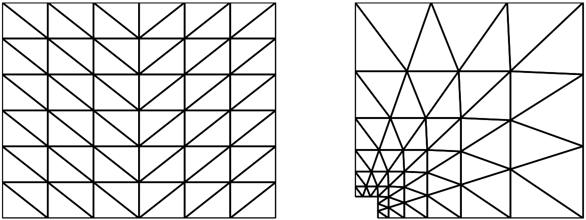


From our favorite mesh generator we have obtained

- Basic mesh data tables, i.e. VX, VY and EToV

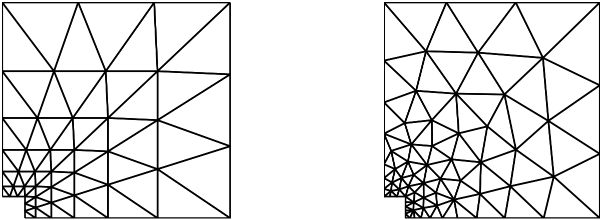
```
VX   = [-1 1 1 -1];
VY   = [-1 -1 1 1];
EToV = [1 2 4;
        2 3 4];
```

Connectivity tables



- The two grids can be described by exactly the same connectivity table!
- The coordinate tables for the vertices are different!
- For simple meshes this can be exploited to generate the connectivity table for a simple mesh and then use it together with a user-defined coordinate table.

Connectivity tables



- However, using a non-uniform triangularization allow for better grid quality and adaptivity for representing the spatial domain.

Mesh generators available

- Lots of standard open source or commercial mesh generation tools available!
  - Test and pick you own favorite!
  - Disadvantage: may require a translation script to be created for use with your own solver.
- Important properties of mesh generators
  - Grid quality (e.g. aspect ratio and element angles)
  - Efficiency
  - Features for handling BCs, adaptivity, etc.
- An example of a free software distribution package for generating unstructured triangular meshes is DistMesh for Matlab.

Translation scripts in Matlab

- Translation scripts take a filename as input and return necessary mesh data as output

```
function [Nv, VX, VY, K, EToV] = MeshReaderGambit2D(FileName)

% function [Nv, VX, VY, K, EToV] = MeshReaderGambit2D(FileName)
% Purpose  : Read in basic grid information to build grid
% NOTE     : gambit(Fluent, Inc) *.neu format is assumed
```

Some useful Matlab commands:

- `fgetl` - read line from file into Matlab string.
- `fscanf`- read formatted data from file.
- `sscanf`- read string under format control.
- `fopen` - open a file for read access.
- `fclose`- close file.

Introduction to DistMesh for Matlab

- Persson, P.-O. and Strang, G. 2004 A simple mesh generator in Matlab. SIAM Review. Download scripts at: http://www-math.mit.edu/~persson/mesh/index.html
- A simple algorithm that combines a physical principle of force equilibrium in a truss structure with a mathematical representation of the geometry using signed distance functions.
- Can generate meshes in 1D, 2D and 3D with few lines of code.

Introduction to DistMesh for Matlab

- Algorithm (Conceptual):
  1. Define a domain using signed distance functions.
  2. Distribute a set of nodes interior to the domain.
  3. Move interior nodes to obtain force equilibrium.
  4. Apply terminate criterion when all nodes are (nearly) fixed in space.
- Post-processing steps (Preparation):
  (Note: not done by DistMesh)
  5. Validate final output!
  6. Reorder element vertices to be defined counter-clockwise (standard convention).
  7. Setup boundary table.
  8. Store mesh for reuse.

Introduction to DistMesh for Matlab

Definition: A signed distance function, $d(x)$

$$d(x) = \begin{cases} < 0 & , x \in \Omega \quad \text{(interior)} \\ 0 & , x \in \partial\Omega \quad \text{(boundary)} \\ > 0 & , x \notin \Omega \quad \text{(exterior)} \end{cases}$$

Define metric using an appropriate norm, e.g. the Euclidian metric.

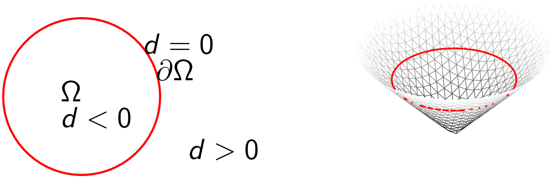
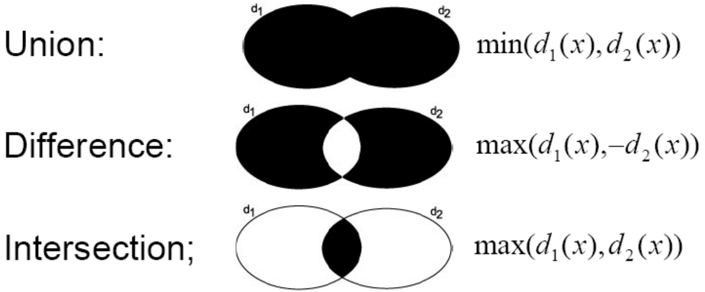


Figure: Example of a signed distance function for a circle.

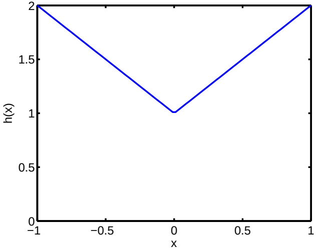
Introduction to DistMesh for Matlab

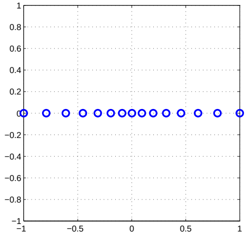
Combine and create geometries defined by distance functions using the Union, difference and intersection operations of sets



Introduction to DistMesh for Matlab

Example 1. Create a nonuniform mesh in 1D with local refinement near center.





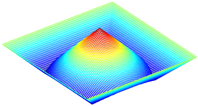
Using DistMesh (in Matlab) only 3 lines of code needed:

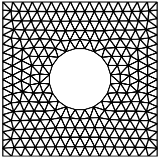
```
>> d=inline('sqrt(sum(p.^2,2))-1','p');
>> h=inline('sqrt(sum(p.^2,2))+1','p');
>> [p,t]=distmeshnd(d,h,0.1,[-1;1],[]);
```

Weight function *h* measures distance from origo and adds a unit to the measure.

Introduction to DistMesh for Matlab

Example 1. Create a uniform mesh for a square with hole.



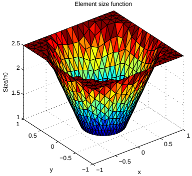


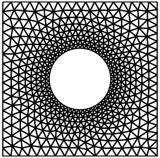
Using DistMesh (in Matlab) only 3 lines of code needed:

```
>> fd=inline('ddiff(drectangle(p,-1,1,-1,1),dcircle(p,0,0,0.4))','p');
>> pfix = [-1,-1;-1,1;1,-1;1,1];
>> [p,t] = distmesh2d(fd,@huniform,0.125,[-1,-1;1,1],pfix);
```

Introduction to DistMesh for Matlab

Example 2. A refined mesh for a square with hole.





Using DistMesh (in Matlab) only 4 lines of code needed:

```
>> fd = inline('ddiff(drectangle(p,-1,1,-1,1),dcircle(p,0,0,0.4))','p')
>> pfix = [-1,-1;-1,1;1,-1;1,1];
>> fh = inline(['min( sqrt( p(:,1).^2 + p(:,2).^2 ) , 1 )'],'p');
>> [p,t] = distmesh2d(fd,fh,0.125/2.5,[-1,-1;1,1],pfix);
```

- Size function `fh` defines *relative* sizes of elements (`fh` constant result in a uniform mesh distribution)
- The *initial* characteristic size of the elements is `h0`.
- In final triangulation, the characteristic size of the smallest elements will be approx. `h0`.

Introduction to DistMesh for Matlab

DistMesh output in two tables;

p Unique vertice coordinates

t Element to Vertice table

(random element orientations by DistMesh)

From these tables we can determine, e.g.

```
>> K=size(t,1);      % Number of elements
>> Nv=size(p,1);     % Number of vertices in mesh
>> Nfaces=size(t,2); % Number of faces/element
>> VX = p(:,1);      % Vertice x-coordinates
>> VY = p(:,2);      % Vertice y-coordinates
>> EToV = t;         % Element to Vertice table
```

To ensure same element node orientations use DistMesh function

```
>> [p,t]=fixmesh(p,t); % remove duplicate nodes and orientate
```

Introduction to DistMesh for Matlab

Example 3. Selecting boundary nodes for a square with hole.

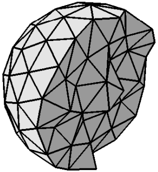
(a) Inner boundary nodes (b) Outer boundary nodes

Nodes can be selected using distance functions; $|d| = 0$ or $|d| <$tol.

```
>> fdInner    = inline(dcircle(p,0,0,0.4),p);
>> nodesInner = find(abs(fdInner([p]))<1e-3);
>> fdOuter    = inline(drectangle(p,-1,1,-1,1),p);
>> nodesOuter = find(abs(fdOuter([p]))<1e-3);
>> nodesB     = find(abs(fd([p]))<1e-3);
```

Introduction to DistMesh for Matlab

Example 4. Uniform mesh for a unit ball (3D).



```
>> fh = @huniform;
>> fd=inline('sqrt(sum(p.^2,2))-1','p'); % ball
>> Bbox = [-1 -1 -1; 1 1 1]; % cube
>> Fix  = [-1 -1 -1; 1 -1 -1; 1 1 -1; -1 1 -1;...
           -1 -1  1; 1 -1  1; 1 1  1; -1 1  1];
>> [Vert,EToV]=distmeshnd(fd,fh,h0,Bbox, Fix);
```

Visualization

MATLAB commands for visualization:

```
% 2-D Triangular plot (also works for quadrilaterals!)
>> triplot(t,p(:,1),p(:,2),'k')

% 3-D Visualization of solution
>> trimesh(t,p(:,1),p(:,2),u)

% 3-D Visualization of solution
>> trisurf(t,p(:,1),p(:,2),u)

% 3-D Visualization of part of solution
>> trisurf(t(idxlist,:),p(:,1),p(:,2),u)

% Visualization of node connections in matrix
>> gplot(A,p)
```

Computing geometric information

We seek to determine outward pointing normal vectors for an element edge of a straight-sided polygon.

Assume that the order of element vertices is counter-clockwise, then for an boundary edge defined from (x_1, y_1) to (x_2, y_2) we find

$$\Delta x = x_2 - x_1, \quad \Delta y = y_2 - y_1$$

and thus a tangential vector becomes

$$\mathbf{t} = (t_1, t_2)^T = (\Delta x, \Delta y)^T$$

which should be orthogonal to the normal vector. Hence an outward pointing normalized vector is given as

$$\mathbf{n} = (n_1, n_2)^T = (t_2, -t_1)^T / \sqrt{t_1^2 + t_2^2}$$

Normal vectors useful for imposing boundary conditions.

Local vertice ordering

To make sure that the vertices are ordered in an counter-clockwise fashion, the following metric can be used

$$D = \begin{pmatrix} x^1 - x^3 \\ y^1 - y^3 \end{pmatrix} \cdot \begin{pmatrix} y^2 - y^3 \\ -(x^2 - x^3) \end{pmatrix} = \hat{\mathbf{t}}_{31} \cdot \hat{\mathbf{n}}_{32}$$

If $D < 0$ then ordering is clockwise and if $D > 0$ counter-clockwise.

```
function [EToV] = Reorder(EToV,VX,VY)
% Purpose: Reorder elements to ensure counter clockwise orientation
x1 = VX(EToV(:,1)); y1 = VY(EToV(:,1));
x2 = VX(EToV(:,2)); y2 = VY(EToV(:,2));
x3 = VX(EToV(:,3)); y3 = VY(EToV(:,3));
D = (x1-x3).*(y2-y3)-(x2-x3).*(y1-y3);
i = find(D<0);
EToV(i,:) = EToV(i,[1 3 2]); % reorder
```

Creating special index maps I

For imposing boundary conditions or extracting information from the solution it can be useful to create special index maps.

Having already created a mesh, create a new boundary table for all element faces

```
>> BCType = int8(not(EToE)); % initialization
```

This table can then be used to store information about different types of boundaries, e.g. Inflow/Outflow, West/East, etc.

Creating special index maps I

To create different index maps for imposing special types of boundary conditions, e.g. Dirichlet and Neumann BC's on all or selected boundaries

```
% for selecting all outer boundaries
>> x1 = -1; x2 = 1; y1 = -1; y2 = 1;
>> fd = @(p) drectangle(p,x1,x2,y1,y2);
>> BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fd,AllBoundaries);

% for selecting south and west boundaries
>> fd = @(p) drectangle(p,-1,2,-1,2);
>> BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fd,Dirichlet);

% select north and east boundaries
>> fd = @(p) drectangle(p,-2,1,-2,1);
>> BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fd,Neuman);
```

Note: new version v2 of CorrectBCTable in ServiceRoutintes/

Creating special index maps I

Then, using the **BCType** table we can create our special index maps

```
% face maps
>> mapB   = ConstructMap(BCType,AllBoundaries);
>> mapD   = ConstructMap(BCType,Dirichlet);
>> mapN   = ConstructMap(BCType,Neuman);

% volume maps
>> vmapB  = vmapM(mapB);
>> vmapN  = vmapM(mapN);
>> vmapD  = vmapM(mapD);
```

Remember to validate the created index maps

Creating special index maps

In problems with periodic boundaries the standard indexmaps can be modified systematically in the following way

1. Create and modify a BCType table to hold information about boundary types
   - `ServiceRoutines/CorrectBCTable_v2`
2. For simple opposing boundaries create a distance function for generating a sorted list of face center distances
3. From the sorted list, create indexmaps for each boundary
   - **ConstructPeriodicMap**
4. Modify volume-to-face index map vmapP to account for periodicity.
5. Validate implementation!

Let's consider a square mesh $\Omega_h([-1,1]^2)$...

Creating special index maps

Step 1: Create and modify a BCType table

```
BCType = int16(not(EToE));

fdW = @(p) drectangle(p,-1,2,-2,2);
fdE = @(p) drectangle(p,-2,1,-2,2);

BCcodeW = 1; BCcodeE = 2;

BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fdW,BCcodeW);
BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fdE,BCcodeE);

fdS = @(p) drectangle(p,-2,2,-1,2);
fdN = @(p) drectangle(p,-2,2,-2,1);

BCcodeS = 3; BCcodeN = 4;

BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fdS,BCcodeS);
BCType = CorrectBCTable_v2(EToV,VX,VY,BCType,fdN,BCcodeN);
```

Creating special index maps

Step 2: Create a distance function useful for sorting opposing face centers

```
pv = [-1 1; 1 -1;];
fd = @(p) dsegment(p,pv); % line segment from (-1,1) to (1,-1)
```

Step 3: Create indexmaps for each periodic boundary

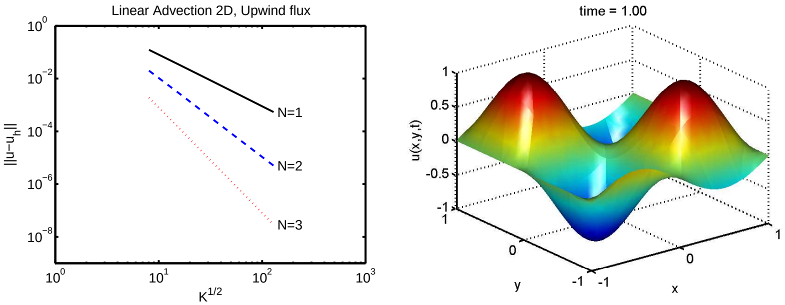
```
[mapW,mapE] = ConstructPeriodicMap(EToV,VX,VY,BCType,BCcodeW,BCcodeE,fd);
[mapS,mapN] = ConstructPeriodicMap(EToV,VX,VY,BCType,BCcodeS,BCcodeN,fd);
```

Step 4: Modify exterior vmapP to be periodic with vmapM

```
vmapP(mapW) = vmapM(mapW);
vmapP(mapE) = vmapM(mapE);
vmapP(mapS) = vmapM(mapS);
vmapP(mapN) = vmapM(mapN);
```

Creating special index maps

Step 5: Validation!



- Periodic initial condition $u_h(x, y, 0)$
- Constant advection speed vector arbitrary $\mathbf{c} = (c_x, c_y)^T$
- Upwind flux gives as expected ideal convergence $\mathcal{O}(h^{N+1})$

Creating special index maps

```
function [rhsu] = AdvecRHS2DupwindPeriodic(u, timelocal, cx, cy, alpha)

% function [rhsu] = AdvecRHS2D(u, timelocal, a, alpha)
% Purpose  : Evaluate RHS flux in 2D advection equation
%            using upwinding

Globals2D;

% Define flux differences at faces
df = zeros(Nfp*Nfaces,K);

% phase speed in normal directions
cn = cx*nx(:) + cy*ny(:);

% upwinding according to characteristics
ustar = 0.5*(cn+abs(cn)).*u(vmapM) + 0.5*(cn-abs(cn)).*u(vmapP);
df(:) = cn.*u(vmapM) - ustar;

% local derivatives of fields
[ux,uy] = Grad2D(u);

% compute right hand sides of the PDE's
rhsu  = -(cx.*ux + cy.*uy) + LIFT*(Fscale.*df);
return
```

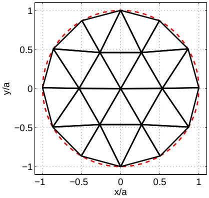
What defines a "good" mesh?

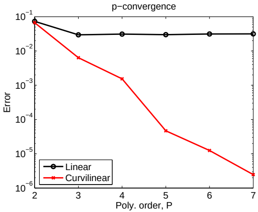
To define a "good" mesh we are usually concerned about

- can we adequately represent the (usually) unknown solution(!)
- using minimal number of elements for minimal cost in solution process for a given numerical accuracy requirement, recall for DG-FEM

$$\mathrm{CPU} \propto C(T)K(N+1)^2, \quad ||u-u_h||_{2,\Omega_h} \propto \mathcal{O}(h^p)$$

- approximating the right geometry of the problem(!)





Three general rules for "good" meshes

There are three general rules dictated by error analysis;

- very large and small element angles should be avoided
  - this suggest that equilateral triangles are optimal
- elements should be placed most densely in regions where the solution of the problem and its derivatives are expected to vary rapidly,
- high accuracy requires a fine mesh or many nodes per element (the latter conditions yields high accuracy, however, only if the solution is sufficiently smooth).

As a user, it is always a good idea to visualize the mesh and to check if these criteria are met.

- To improve mesh quality, it can be beneficial to apply some mesh smoothing procedure
  Fx. use **smoothmesh.m** from Mesh2D v23, Matlab Central Exchange.

A simple measure of mesh quality

The mesh quality measure described by Persson & Strang (2004) is adopted in the following.

A common mesh quality measure is the following ratio

$$q = 2\frac{r_{in}}{r_{out}}$$

where r_{in} is the radius of the largest inscribed circle and r_{out} is the smallest circumscribed circle.

- Equilateral triangles has $q = 1$
- Degenerate triangles has $q = 0$
- "Good triangles" we define as having $q > 0.5$ (rule of thumb)

Laplacian smoothing

To improve the mesh quality, we can apply a simple Laplacian smoothing procedure

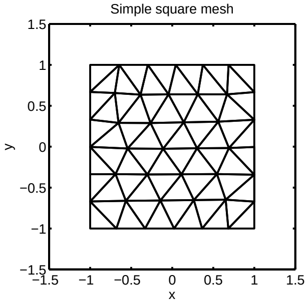
$$\mathbf{x}_i^{[k+1]} = \frac{1}{N_{i,connect}} \sum_{j=1}^{N_{i,connect}} \alpha_i \mathbf{x}_j^{[k]}, \quad \forall i : \mathbf{x}_i \notin \partial\Omega_h$$

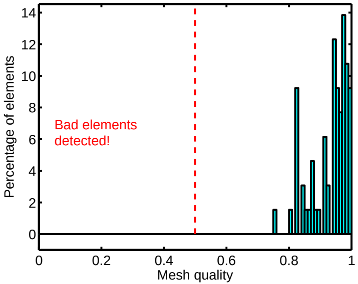
with α_j weight factors and $N_{i,connect}$ the number of nodes connected to the i'th node dictated by the mesh structure.

There are a few pitfalls

- Mesh tangling can occur near reentrant corners and needs special treatment.
- Local mesh adaption (anisotropic mesh density) can be reduced in the process.

Mesh quality

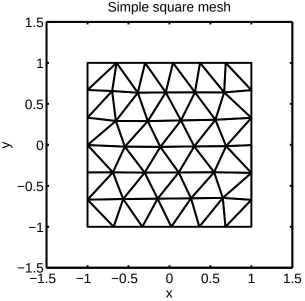


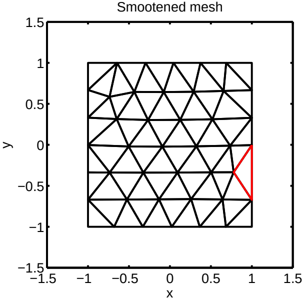



```
fd    = inline('drectangle(p,-1,1,-1,1)','p','param');
fh    = @huniform;
h0    = 0.35;
Bbox  = [-1 -1; 1 1];
pfix  = [-1 -1; 1 -1; 1 1; -1 1];
param = [];

% Call distmesh
[Vert,EToV]=distmesh2d(fd,fh,h0,Bbox,pfix,param);
[q] = MeshQuality(EToV,VX,VY);
```

Mesh quality



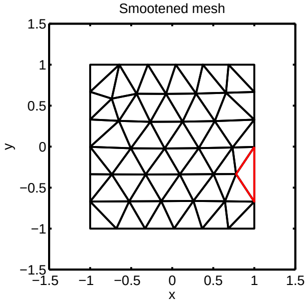


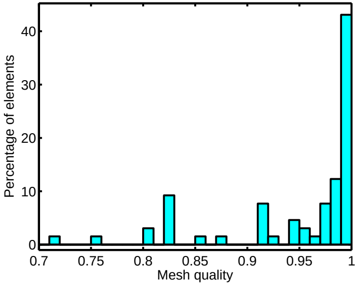
Remark: Degenerated triangle fixed by Laplacian smoothing.

```
% Call distmesh
[Vert,EToV]=distmesh2d(fd,fh,h0,Bbox,pfix,param);

% Call Mesh2d v23 function
maxit = 100; tol = 1e-10;
[p,EToV] = smoothmesh([VX' VY'],EToV,maxit,tol);
VX = p(:,1)'; VY = p(:,2)';
```

Mesh quality





Open source mesh software

Unstructured mesh generation software

- DistMesh (http://www-math.mit.edu/~persson/mesh/)
- Triangle (http://www.cs.cmu.edu/~quake/triangle.html)
- Mesh2D (http://www.mathworks.com/matlabcentral/)
- Gmsh (http://www.geuz.org/gmsh/)

Note: list is not exhaustive.

Do you have a favorite?